MOTOROLA
Semiconductor Products Inc.

AN-852
Application Note

MONITOR FOR THE MC146805G2L1 MICROCOMPUTER

Prepared by
David Bush
Microprocessor Product Engineer
and
Ed Rupp
Microprocessor System Design Engineer
Austin, Texas

INTRODUCTION

The MC146805G2 is a fully static single-chip CMOS Microcomputer. It has 112 bytes of RAM, 2106 bytes of user ROM, four 8-bit input/output ports, a timer, and an on-chip oscillator. The MC146805G2L1 ROM contains a monitor routine which provides the user with the ability to evaluate the MC146805G2 using a standard RS232 terminal. The user can enter short programs into the on-chip RAM and execute them via the monitor. A description of the monitor operation follows along with an assembled listing of the actual program.

MONITOR MODE

In this mode the MC146805G2L1 Microcomputer is connected to a terminal capable of running at 300, 1200, 4800, or 9600 baud. Figure 1 contains a schematic diagram of the monitor mode connections and a table showing C0 and C1 switch settings to obtain a baud rate that matches the terminal. Be sure the oscillator frequency is 3.579545 MHz. Any area of RAM from location $18 to $7A may be used for program storage; however, upper locations may be needed for user stack.

When the microcomputer is reset, a power-up message is printed. Following the message, the prompt character "." is printed and the monitor waits for a response. The response may consist of single letter commands with some commands requiring additional input. Unrecognized commands respond by printing "?". Valid commands are:

R — Display the Register
A — Display/Change the Accumulator
X — Display/Change the Index Register
M — Display/Change Memory
C — Continue Program Execution
E — Execute Program at Address
S — Display State of I/O and Timer

R — Display the Register

The processor registers are displayed as they appear on the stack. The format of the register print is:

HINZC AA XX PP

The first field shows the state of the condition code register bits. Each bit in the register has a single letter corresponding to the bit name. If the letter is present, the bit is 1. If a "." is printed in place of the letter, that bit is 0. For example, "H..ZC" means that the H, Z, and C bits are 1 and that the I and N bits are 0. The remainder of the line shows the status of the accumulator, index register, and program counter, respectively. The stack pointer is always at a fixed address (in this case $7A). The values shown are the values loaded into the CPU when a "C" or "E" command is executed. All register values except the condition code register can be changed with other commands. To change the condition code register, it is necessary to use the memory change command and modify location $7B.

A — Examine/Change the Accumulator

This command begins by printing the current value of the accumulator and then waits for more input. In order to change the current value, type in a new value (two hex digits). To leave the accumulator unchanged, type any non-hex digit (a space is a good choice).

X — Examine/Change the Index Register

This procedure is the same as the "A" command, but affects the index register instead.

M — Examine/Change Memory

Any memory location may be examined or changed with this command (except of course, ROM). To begin, type "M" followed by a hexadecimal address in the range $0000-$1FFF. The monitor responds by beginning a new line

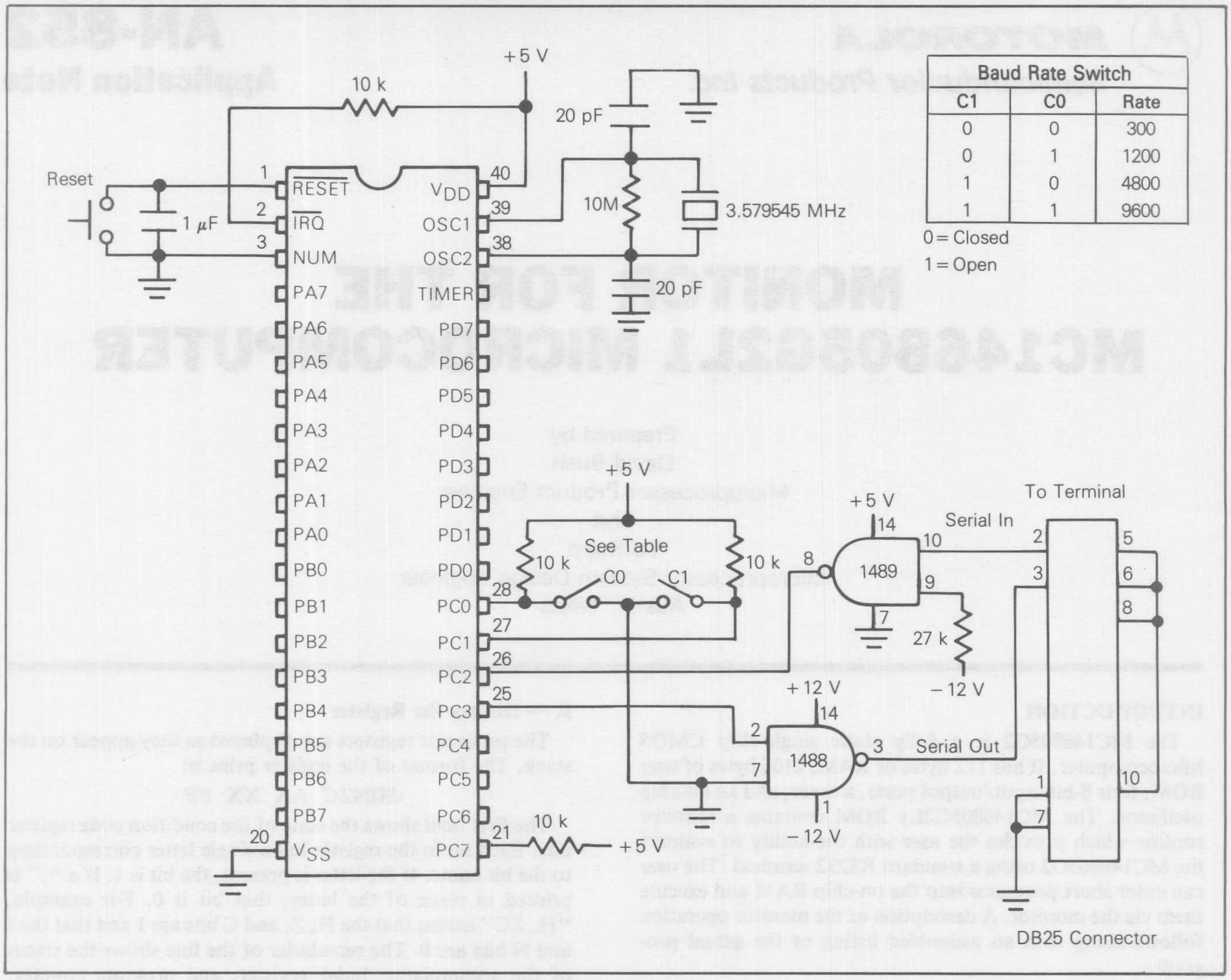

FIGURE 1. Monitor Mode Schematic Diagram

and printing the memory address followed by the current contents of that location. At this point you may type:

1. "." and re-examine the same byte. (Try this with location $0008.)
2. "∧" and go to the previous byte. Typing "∧" at location $0000 causes the monitor to go to $1FFF.
3. "CR" and go to the next byte. "CR" is the carriage return character. The byte after $1FFF is $0000.
4. "DD", where "DD" is a valid 2-digit hexadecimal number. The new data is stored at the current address and the monitor then goes to the next location. This means that to enter a program it is only necessary to go to the starting address of the program and start typing in the bytes. To see if the byte was really inputted, you can use the "∧" character to return to the last byte typed in.
5. Finally, any character other than those described above causes the memory command to return to the prompt level of the monitor and prints "."

C — Continue Program Execution

The "C" command merely executes an RTI instruction. This means that all the registers are reloaded exactly as they are shown in the register display. Execution continues until the reset switch is depressed or the processor executes an SWI. Upon executing an SWI, the monitor regains control and prints the prompt character. This feature can be used for an elementary form of breakpoints. Since there is really no way to know where the stack pointer is after an SWI, the monitor assumes that it is at $7A. This will not be the case if an SWI is part of a subroutine. In this case, the monitor will be re-entered but the stack pointer will point to $78. This is perfectly valid and typing "C" will pick up the program from where it left off. However, the A, X, R, and E commands all assume the stack starts at $7A and will not function properly. If the stack location is known, it is still possible to examine the registers by using the M command.

E — Start Execution at Address

The "E" command waits for a valid memory address

($0000-$1FFF) and places the address typed on the stack at locations $7E and $7F. The command then executes an RTI just like the "C" command. If the address typed is not a valid memory address, the command exits to the monitor without changing the current program counter value.

S — Display I/O States and Timer

The "S" command displays ports A, B, C, and D data along with the timer data and control register contents. The format of the display is:

A B C D TIM TCR

The data displayed is simply memory (RAM) locations $0000-$0003 with $0008 and $0009. Ports A, B, and D may be written to by first making them all outputs; i.e., for port A, change location $0004 (port A DDR) to $FF. Port C and the timer registers cannot be changed as they are used by the monitor.

MONITOR PROGRAM

A flowchart for the monitor mode program is provided in Figure 2. A listing for the ROM monitor program is attached to the end of this application note.

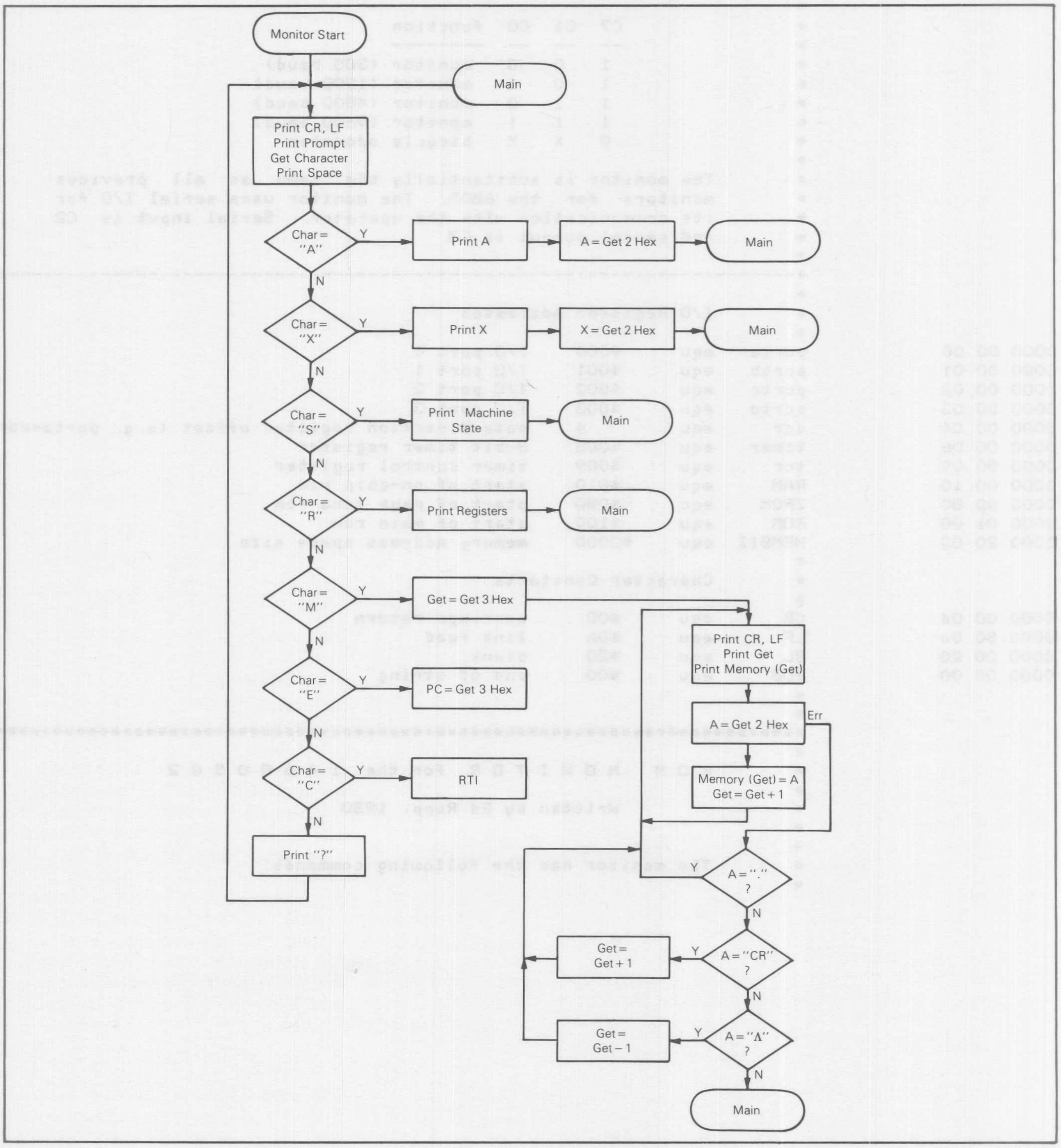

FIGURE 2. Monitor Mode Operating Flowchart

```
                        *
                        *         M C 1 4 6 8 0 5 G 2   R O M   P A T T E R N
                        *
                        *       The MC6805G2 single-chip microcomputer is a  40-pin  CMOS
                        *       device  with  2096  bytes  of ROM, 112 bytes of RAM, four
                        *       8-bit I/O  ports,  a  timer  and  an  external  interrupt
                        *       input   The  ROM contains two separate programs.  Either
                        *       of these programs may be selected on reset by wiring port
                        *       C as follows:
                        *
                        *             C7  C1  C0  function
                        *             --  --  --  --------
                        *              1   0   0   monitor (300 baud)
                        *              1   0   1   monitor (1200 baud)
                        *              1   1   0   monitor (4800 baud)
                        *              1   1   1   monitor (9600 baud)
                        *              0   X   X   bicycle odometer
                        *
                        *       The monitor is substantially the  same  as  all  previous
                        *       monitors  for  the 6805.  The monitor uses serial I/O for
                        *       its communication with the operator.  Serial input is  C2
                        *       and serial output is C3.
                        *
                        *--------------------------------------------------------------------------
                        *
                        *       I/O Register Addresses
                        *
0000 00 00              porta   equ     $000    I/O port 0
0000 00 01              portb   equ     $001    I/O port 1
0000 00 02              portc   equ     $002    I/O port 2
0000 00 03              portd   equ     $003    I/O port 3
0000 00 04              ddr     equ        4    data direction register offset (e.g. porta+ddr)
0000 00 08              timer   equ     $008    8-bit timer register
0000 00 09              tcr     equ     $009    timer control register
0000 00 10              RAM     equ     $010    start of on-chip ram
0000 00 80              ZROM    equ     $080    start of page zero rom
0000 01 00              ROM     equ     $100    start of main rom
0000 20 00              MEMSIZ  equ    $2000    memory address space size
                        *
                        *       Character Constants
                        *
0000 00 0d              CR      equ     $0D     carriage return
0000 00 0a              LF      equ     $0A     line feed
0000 00 20              BL      equ     $20     blank
0000 00 00              EOS     equ     $00     end of string
                        *
                        *
                        **************************************************************************
                        *
                        *       R O M   M O N I T O R   for the  1 4 6 8 0 5 G 2
                        *
                        *             Written by Ed Rupp, 1980
                        *
                        *
                        *
                        *       The monitor has the following commands:
                        *
```

```
                          *         R --  Print registers.
                          *               format is CCCCC AA XX PPP
                          *
                          *         A --  Print/change A accumulator.
                          *               Prints the register value, then
                          *               waits for new value.  Type
                          *               any non-hex character to exit.
                          *
                          *         X --  Print/change X accumulator.
                          *               Works the same as 'A', except modifies X instead.
                          *
                          *         M --  Memory examine/change.
                          *               Type M AAA to begin,
                          *               then type: .  -- to re-examine current
                          *                          ^  -- to examine previous
                          *                          CR -- to examine next
                          *                          DD -- new data
                          *                  Anything else exits memory command.
                          *
                          *         C --  Continue program.  Execution starts at
                          *                  the location specified in the program
                          *                  counter, and
                          *                  continues until an  swi is executed
                          *                  or until reset.
                          *
                          *         E --  Execute from address.  Format is
                          *                  E AAAA.  AAAA is any valid memory address.
                          *
                          *         S --  Display Machine State.  All important registers are
                          *                  displayed.
                          *
                          *         Special Equates
                          *
0602 00 2e                PROMPT    equ     '.          prompt character
0602 00 0d                FWD       equu    CR          go to next byte
0602 00 5e                BACK      equ     '^          go to previous byte
0602 00 2e                SAME      equ     '.          re-examine same byte
                          *
                          *         Other
                          *
0602 00 7f                initsp    equ     $7F         initial stack pointer value
0602 00 7a                stack     equ     initsp-5    top of stack
                          *
                          *         ram variables
                          *
0602 00 10                get       equ     RAM+0       4-byte no-mans land, see pick and drop subroutines
0602 00 14                atemp     equ     RAM+4       acca temp for getc,putc
0602 00 15                xtemp     equ     RAM+5       x reg. temp for getc,putc
0602 00 16                char      equ     RAM+6       current input/output character
0602 00 17                count     equ     RAM+7       number of bits left to get/send
                          *
                          *         state --- print machine state
                          *
                          *           A  B  C  D TIM TCR
                          *          dd dd dd dd  dd  dd
```

```
                              *
                              *      header string for I/O register display
                              *
0602 0d 0a                    iomsg   fcb      CR,LF
0604 20 41 20 20 42 20                fcc      / A  B  C  D TIM TCR/
     20 43 20 20 44 20
     54 49 4d 20 54 43
     52
0617 0d 0a 00                         fcb      CR,LF,EOS
                              *
061a 5f                       state   clrx
061b d6 06 02                 state2  lda      iomsg,x get next char
061e a1 00                            cmp      #EOS    quit?
0620 27 06                            beq      state3  yes, now print values
0622 cd 08 01                         jsr      putc    no, print char
0625 5c                               incx             bump pointer
0626 20 f3                            bra      state2  do it again
0628                          state3
                              *
                              *      now print values underneath the header
                              *
0628 5f                               clrx
0629 f6                       pio     lda      ,x      start with I/O ports
062a cd 07 5e                         jsr      putbyt
062d cd 07 8b                         jsr      puts
0630 5c                               incx
0631 a3 04                            cpx      #4      end of I/O?
0633 26 f4                            bne      pio     no, do more
                              *
0635 cd 07 8b                         jsr      puts
0638 b6 08                            lda      timer   now print the value in the timer
063a cd 07 5e                         jsr      putbyt
063d cd 07 8b                         jsr      puts
0640 cd 07 8b                         jsr      puts
0643 b6 09                            lda      tcr     the control register too
0645 cd 07 5e                         jsr      putbyt
0648 20 48                            bra      monit   all done
                              *
                              *      pcc --- print condition codes
                              *
                              *      string for pcc subroutine
                              *
064a 48 49 4e 5a 43           ccstr   fcc      /HINZC/
                              *
064f b6 7b                    pcc     lda      stack+1 condition codes in acca
0651 48                               asla             move h bit to bit 7
0652 48                               asla
0653 48                               asla
0654 b7 10                            sta      get     save it
0656 5f                               clrx
0657 a6 2e                    pcc2    lda      #'.
0659 38 10                            asl      get     put bit in c
065b 24 03                            bcc      pcc3    bit off means print .
065d d6 06 4a                         lda      ccstr,x pickup appropriate character
0660 cd 08 01                 pcc3    jsr      putc    print . or character
0663 5c                               incx             point to next in string
```

```
0664 a3 05                  cpx     #5          quit after printing all 5 bits
0666 25 ef                  blo     pcc2
0668 81                     rts
                    *
                    *       seta --- examine/change accumulator A
                    *
0669 ae 7c          seta    ldx     #stack+2 point to A
066b 20 02                  bra     setany
                    *
                    *       setx --- examine/change accumulator X
                    *
066d ae 7d          setx    ldx     #stack+3 point to X
                    *
                    *       setany --- print (x) and change if necessary
                    *
066f f6             setany  lda     ,x          pick up the data, and
0670 cd 07 5e               jsr     putbyt  print it
0673 cd 07 8b               jsr     puts
0676 cd 07 94               jsr     getbyt  see if it should be changed
0679 25 17                  bcs     monit       error, no change
067b f7                     sta     ,x          else replace with new value
067c 20 14                  bra     monit         now return
                    *
                    *       regs --- print cpu registers
                    *
067e ad cf          regs    bsr     pcc         print cc register
0680 cd 07 8b               jsr     puts        separate from next stuff
0683 3f 11                  clr     get+1       point to page zero,
0685 a6 7c                  lda     #stack+2
0687 b7 12                  sta     get+2
0689 cd 07 4b               jsr     out2hs      continue print with A
068c cd 07 4b               jsr     out2hs      X and finally the
068f cd 07 43               jsr     out4hs      Program Counter
                    *
                    *         fall into main loop
                    *
                    *         monit --- print prompt and decode commands
                    *
0692 cd 07 7d       monit   jsr     crlf        go to next line
0695 a6 2e                  lda     #PROMPT
0697 cd 08 01               jsr     putc        print the prompt
069a cd 07 c3               jsr     getc        get the command character
069d a4 7f                  and     #%1111111 mask parity
069f cd 07 8b               jsr     puts        print space (won't destroy A)
06a2 a1 41                  cmp     #'A         change A
06a4 27 c3                  beq     seta
06a6 a1 58                  cmp     #'X         change X
06a8 27 c3                  beq     setx
06aa a1 52                  cmp     #'R         registers
06ac 27 d0                  beq     regs
06ae a1 45                  cmp     #'E         execute
06b0 27 16                  beq     exec
06b2 a1 43                  cmp     #'C         continue
06b4 27 21                  beq     cont
06b6 a1 4d                  cmp     #'M         memory
06b8 27 1e                  beq     memory
```

```
06ba a1 53            cmp     #'S      display machine state
06bc 26 03            bne     monit2
06be cc 06 1a         jmp     state    commands are getting too far away
                *
06c1 06 c1      monit2 equ    *
06c1 a6 3f            lda     #'?      none of the above
06c3 cd 08 01         jsr     putc
06c6 20 ca            bra     monit    loop around
                *
                *     exec --- execute from given address
                *
06c8 cd 07 94   exec  jsr     getbyt   get high nybble
06cb 25 c5            bcs     monit     bad digit
06cd 97               tax               save for a second
06ce cd 07 94         jsr     getbyt   now the low byte
06d1 25 bf            bcs     monit     bad address
06d3 b7 7f            sta     stack+5 program counter low
06d5 bf 7e            stx     stack+4 program counter high
                *     cont --- continue users program
                *
06d7 80         cont  rti              simple enough
                *
                *     memory --- memory examine/change
                *
06d8 cd 07 94   memory jsr    getbyt   build address
06db 25 b5            bcs     monit    bad hex character
06dd b7 11            sta     get+1
06df cd 07 94         jsr     getbyt
06e2 25 ae            bcs     monit    bad hex character
06e4 b7 12            sta     get+2    address is now in get+1&2
06e6 cd 07 7d   mem2  jsr     crlf     begin new line
06e9 b6 11            lda     get+1    print current location
06eb a4 1f            and     #$1F     mask upper 3 bits (8K map)
06ed cd 07 5e         jsr     putbyt
06f0 b6 12            lda     get+2
06f2 cd 07 5e         jsr     putbyt
06f5 cd 07 8b         jsr     puts     a blank, then
06f8 ad 2c            bsr     pick     get that byte
06fa cd 07 5e         jsr     putbyt   and print it
06fd cd 07 8b         jsr     puts     another blank,
0700 cd 07 94         jsr     getbyt   try to get a byte
0703 25 06            bcs     mem3     might be a special character
0705 ad 25            bsr     drop     otherwise, put it and continue
0707 ad 33      mem4  bsr     bump     go to next address
0709 20 db            bra     mem2     and repeat
070b a1 2e      mem3  cmp     #SAME    re-examine same?
070d 27 d7            beq     mem2     yes, return without bumping
070f a1 0d            cmp     #FWD     go to next?
0711 27 f4            beq     mem4     yes, bump then loop
0713 a1 5e            cmp     #BACK    go back one byte?
0715 26 0c            bne     xmonit   no, exit memory command
0717 3a 12            dec     get+2    decrement low byte
0719 b6 12            lda     get+2    check for underflow
071b a1 ff            cmp     #$FF
071d 26 c7            bne     mem2     no underflow
```

```
071f 3a 11                  dec     get+1
0721 20 c3                  bra     mem2
                    *
                    *       convenient transfer point back to monit
                    *
0723 cc 06 92       xmonit  jmp     monit   return to monit
                    *
                    *       utilities
                    *
                    *       pick --- get byte from anywhere in memory
                    *                this is a horrible routine (not merely
                    *                self-modifying, but self-creating)
                    *
                    *       get+1&2 point to address to read,
                    *       byte is returned in A
                    *       X is unchanged at exit
0726 bf 15          pick    stx     xtemp   save X
0728 ae d6                  ldx     #$D6    D6=lda 2-byte indexed
072a 20 04                  bra     common
                    *
                    *
                    *       drop --- put byte to any memory location.
                    *                has the same undesirable properties
                    *                as pick
                    *       A has byte to store, and get+1&2 points
                    *       to location to store
                    *       A and X unchanged at exit
                    *
072c bf 15          drop    stx     xtemp   save X
072e ae d7                  ldx     #$D7    d7=sta 2-byte indexed
                    *
                    *
0730 bf 10          common  stx     get     put opcode in place
0732 ae 81                  ldx     #$81    81=rts
0734 bf 13                  stx     get+3   now the return
0736 5f                     clrx            we want zero offset
0737 bd 10                  jsr     get     execute this mess
0739 be 15                  ldx     xtemp   restore X
073b 81                     rts             and exit
                    *
                    *       bump --- add one to current memory pointer
                    *
                    *       A and X unchanged
                    *
073c 3c 12          bump    inc     get+2   increment low byte
073e 26 02                  bne     bump2   non-zero means  no carry
0740 3c 11                  inc     get+1   increment high nybble
0742 81             bump2   rts
                    *
                    *
                    *       out4hs --- print word pointed to as an address, bump pointer
                    *                  X is unchanged at exit
                    *
0743 ad e1          out4hs  bsr     pick    get high nybble
0745 a4 1f                  and     #$1F    mask high bits
```

```
0747 ad 15                   bsr     putbyt  and print it
0749 ad f1                   bsr     bump    go to next address
                   *
                   *         out2hs --- print byte pointed to, then a space. bump pointer
                   *                    X is unchanged at exit
                   *
074b ad d9         out2hs    bsr     pick    get the byte
074d b7 10                   sta     get     save A
074f 44                      lsra
0750 44                      lsra
0751 44                      lsra
0752 44                      lsra            shift high to low
0753 ad 16                   bsr     putnyb
0755 b6 10                   lda     get
0757 ad 12                   bsr     putnyb
0759 ad e1                   bsr     bump    go to next
075b ad 2e                   bsr     puts    finish up with a blank
075d 81                      rts
                   *
                   *         putbyt --- print A in hex
                   *                    A and X unchanged
                   *
075e b7 10         putbyt    sta     get     save A
0760 44                      lsra
0761 44                      lsra
0762 44                      lsra
0763 44                      lsra            shift high nybble down
0764 ad 05                   bsr     putnyb  print it
0766 b6 10                   lda     get
0768 ad 01                   bsr     putnyb  print low nybble
076a 81                      rts
                   *
                   *         putnyb --- print lower nybble of A in hex
                   *                    A and X unchanged, high nybble
                   *                    of A is ignored.
                   *
076b b7 13         putnyb    sta     get+3   save A in yet another temp
076d a4 0f                   and     #$F     mask off high nybble
076f ab 30                   add     #'0     add ascii zero
0771 a1 39                   cmp     #'9     check for A-F
0773 23 02                   bls     putny2
0775 ab 07                   add     #'A-'9-1 adjustment for hex A-F
0777 cd 08 01      putny2    jsr     putc
077a b6 13                   lda     get+3   restore A
077c 81                      rts
                   *
                   *         crlf --- print carriage return, line feed
                   *                  A and X unchanged
                   *
077d b7 10         crlf      sta     get     save
077f a6 0d                   lda     #CR
0781 cd 08 01                jsr     putc
0784 a6 0a                   lda     #LF
0786 ad 79                   bsr     putc
0788 b6 10                   lda     get     restore
078a 81                      rts
```

```
                        *
                        *       puts --- print a blank (space)
                        *               A and X unchanged
                        *
078b b7 10              puts    sta     get     save
078d a6 20                      lda     #BL
078f ad 70                      bsr     putc
0791 b6 10                      lda     get     restore
0793 81                         rts
                        *
                        *       getbyt --- get a hex byte from terminal
                        *
                        *       A gets the byte typed if it was a valid hex number,
                        *       otherwise A gets the last character typed.  The c-bit is
                        *       set on non-hex characters;  cleared  otherwise.  X
                        *       unchanged in any case.
                        *
0794 ad 0f              getbyt  bsr     getnyb  build byte from 2 nybbles
0796 25 0c                      bcs     nobyt   bad character in input
0798 48                         asla
0799 48                         asla
079a 48                         asla
079b 48                         asla            shift nybble to high nybble
079c b7 10                      sta     get     save it
079e ad 05                      bsr     getnyb  get low nybble now
07a0 25 02                      bcs     nobyt   bad character
07a2 bb 10                      add     get     c-bit cleared
07a4 81                 nobyt   rts
                        *
                        *       getnyb --- get hex nybble from terminal
                        *
                        *       A gets the nybble typed if it was in the range 0-F,
                        *       otherwise A gets the character typed.  The c-bit is set
                        *       on non-hex characters;  cleared  otherwise.  X is
                        *       unchanged.
                        *
07a5 ad 1c              getnyb  bsr     getc    get the character
07a7 a4 7f                      and     #%1111111 mask parity
07a9 b7 13                      sta     get+3   save it just in case
07ab a0 30                      sub     #'0     subtract ascii zero
07ad 2b 10                      bmi     nothex  was less than '0'
07af a1 09                      cmp     #9
07b1 23 0a                      bls     gotit
07b3 a0 07                      sub     #'A-'9-1 funny adjustment
07b5 a1 0f                      cmp     #$F     too big?
07b7 22 06                      bhi     nothex  was greater than 'F'
07b9 a1 09                      cmp     #9        check between 9 and A
07bb 23 02                      bls     nothex
07bd 98                 gotit   clc             c=0 means good hex char
07be 81                         rts
07bf b6 13              nothex  lda     get+3   get saved character
07c1 99                         sec
07c2 81                         rts             return with error
                        *
                        *       S e r i a l    I / O    R o u t i n e s
                        *
```

```
                        *       These subroutines are modifications of the original NMOS
                        *       version.  Differences are due to the variation in cycle
                        *       time of CMOS instructions vs. NMOS.
                        *
                        *       Since the INT and TIMER interrupt vectors are used in the
                        *       bicycle odometer, the I-bit should  always  be  set  when
                        *       running  the  monitor.  Hence, the code that fiddles with
                        *       the I-bit has been eliminated.
                        *
                        *
                        *       Definition of serial I/O lines
                        *
                        *       Note: changing 'in' or 'out' will necessitate changing the
                        *       way 'put' is setup during reset.
                        *
07c3 00 02              put     equ     portc   serial I/O port
07c3 00 02              in      equ     2       serial input line#
07c3 00 03              out     equ     3       serial output line#
                        *
                        *       getc --- get a character from the terminal
                        *
                        *       A gets the character typed, X is unchanged.
                        *
07c3 bf 15              getc    stx     xtemp   save X
07c5 a6 08                      lda     #8      number of bits to read
07c7 b7 17                      sta     count
07c9 04 02 fd           getc4   brset   in,put,getc4 wait for hilo transition
                        *
                        *       delay 1/2 bit time
                        *
07cc b6 02                      lda     put
07ce a4 03                      and     #%11    get current baud rate
07d0 97                         tax
07d1 de 08 4b                   ldx     delays,x get loop constant
07d4 a6 04              getc3   lda     #4
07d6 9d                 getc2   nop
07d7 4a                         deca
07d8 26 fc                      bne     getc2
07da 5d                         tstx            loop padding
07db 14 02                      bset    in,put  ditto
07dd 14 02                      bset    in,put  CMOS ditto
07df 5a                         decx
07e0 26 f2                      bne     getc3   major loop test
                        *
                        *       now we should be in the middle of the start bit
                        *
07e2 04 02 e4                   brset   in,put,getc4 false start bit test
07e5 7d                         tst     ,x      more timing delays
07e6 7d                         tst     ,x
07e7 7d                         tst     ,x
                        *
                        *       main loop for getc
                        *
07e8 ad 46              getc7   bsr     delay   (6) common delay routine
07ea 05 02 00                   brclr   in,put,getc6 (5)  test input and set c-bit
07ed 7d                 getc6   tst     ,x      (4) timing equalizer
```

```
07ee 9d                    nop            (2) CMOS equalization
07ef 9d                    nop            (2) CMOS equalization
07f0 9d                    nop            (2) CMOS equalization
07f1 9d                    nop            (2) CMOS equalization
07f2 9d                    nop            (2) CMOS equalization
07f3 9d                    nop            (2) CMOS equalization
07f4 36 16                 ror    char    (5) add this bit to the byte
07f6 3a 17                 dec    count   (5)
07f8 26 ee                 bne    getc7   (3) still more bits to get(see?)
                   *
07fa ad 34                 bsr    delay   wait out the 9th bit
07fc b6 16                 lda    char    get assembled byte
07fe be 15                 ldx    xtemp   restore x
                   *
0800 81                    rts            and return
                   *
                   *       putc --- print a on the terminal
                   *
                   *       X and A unchanged
                   *
0801 b7 16         putc    sta    char
0803 b7 14                 sta    atemp   save it in both places
0805 bf 15                 stx    xtemp   don't forget about X
0807 a6 09                 lda    #9      going to put out
0809 b7 17                 sta    count   9 bits this time
080b 5f                    clrx           for very obscure reasons
080c 98                    clc            this is the start bit
080d 20 02                 bra    putc2   jump in the middle of things
                   *
                   *       main loop for putc
                   *
080f 36 16         putc5   ror    char    (5) get next bit from memory
0811 24 04         putc2   bcc    putc3   (3) now set or clear port bit
0813 16 02                 bset   out,put
0815 20 04                 bra    putc4
0817 17 02         putc3   bclr   out,put (5)
0819 20 00                 bra    putc4   (3) equalize timing again
081b dd 08 30      putc4   jsr    delay,x (7) must be 2-byte indexed jsr
                   *                          this is why X must be zero
081e 43                    coma           (3) CMOS equalization
081f 43                    coma           (3) CMOS equalization
0820 43                    coma           (3) CMOS equalization
0821 3a 17                 dec    count   (5)
0823 26 ea                 bne    putc5   (3) still more bits
                   *
0825 14 02                 bset   in,put   7 cycle delay
0827 16 02                 bset   out,put send stop bit
                   *
0829 ad 05                 bsr    delay   delay for the stop bit
082b be 15                 ldx    xtemp   restore X and
082d b6 14                 lda    atemp   of course A
082f 81                    rts
                   *
                   *       delay --- precise delay for getc/putc
                   *
0830 b6 02         delay   lda    put     first, find out
```

```
0832 a4 03              and     #%11     what the baud rate is
0834 97                 tax
0835 de 08 4b           ldx     delays,x loop constant from table
0838 a6 f8              lda     #$F8     funny adjustment for subroutine overhead
083a ab 09      del3    add     #$09
083c            del2
083c 9d                 nop              CMOS equalization
083d 4a                 deca
083e 26 fc              bne     del2
0840 5d                 tstx             loop padding
0841 14 02              bset    in,put   ditto
0843 14 02              bset    in,put   CMOS ditto
0845 5a                 decx
0846 26 f2              bne     del3     main loop
0848 9d                 nop              CMOS equalization
0849 9d                 nop              CMOS equalization
084a 81                 rts              with X still equal to zero
                *
                *       delays for baud rate calculation
                *
                *       This table must not be put on page zero since
                *       the accessing must take 6 cycles.
                *
084b 20         delays  fcb     32        300 baud
084c 08                 fcb     8        1200 baud
084d 02                 fcb     2        4800 baud
084e 01                 fcb     1        9600 baud
                *
                *     reset --- power on reset routine
                *
                *       Based on a port bit, run the bicycle odometer or the monitor.
                *
084f            reset
084f 0e 02 03           brset   7,portc,other
0852 cc 01 54           jmp     odo      be a bicycle odometer
                *
                *       run the monitor
                *
0855            other
0855 a6 08              lda     #%1000   setup port for serial io
0857 b7 02              sta     put      set output to mark level
0859 b7 06              sta     put+ddr  set ddr to have one output
                *
                *       print sign-on message
                *
085b 5f                 clrx
085c d6 08 6c   babble  lda     msg,x    get next character
085f a1 00              cmp     #EOS     last char?
0861 27 06              beq     mstart   yes, start monitor
0863 cd 08 01           jsr     putc     and print it
0866 5c                 incx             advance to next char
0867 20 f3              bra     babble   more message
0869            mstart
0869 83                 swi              push machine state and go to monitor routine
086a 20 e3              bra       reset  loop around
                *
```

```
                            *       msg --- power up message
                            *
086c 0d 0a                  msg     fcb     CR,LF
086e 31 34 36 38 30 35              fcc     /146805G2/
     47 32
0876 00                             fcb     EOS
                            *
                            *
                            ******************************************************************
                            *
                            *       interrupt vectors
                            *
1ff6                                org     MEMSIZ-10 start of vectors
                            *
1ff6 01 e0                          fdb     onemil  exit wait state       \
1ff8 01 e0                          fdb     onemil  timer interrupt        |- odometer vectors
1ffa 02 46                          fdb     wheel   external interrupt    /
1ffc 06 92                          fdb     monit   swi to main entry point
1ffe 08 4f                          fdb     reset   power on vector
```

MOTOROLA *Semiconductor Products Inc.*

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721 • A SUBSIDIARY OF MOTOROLA INC.

A14957 PRINTED IN USA 11-81 IMPERIAL LITHO C01419 18,000

AN 852